Savitribai Phule Pune University

S.Y.BCS (Computer Science) Practical Examination (2019 Pattern)

Lab Course 234 SEM IV
Data Structure Laboratory

Duration: 3 Hours Maximum **Marks: 35**

Q 1. Write a C program that accepts the vertices and edges of a graph and stores it as an adjacency matrix. Display the adjacency matrix. [15 Marks]

```
#include <stdio.h>

// N vertices and M Edges
int N, M;

// Function to create Adjacency Matrix
void createAdjMatrix(int Adj[][N + 1],int arr[][2])
{

	// Initialise all value to this
	// Adjacency list to zero
	for (int i = 0; i < N + 1; i++) {

		for (int j = 0; j < N + 1; j++) {
			Adj[i][j] = 0;
		}
	}

	// Traverse the array of Edges
	for (int i = 0; i < M; i++) {

		// Find X and Y of Edges
		int x = arr[i][0];
		int y = arr[i][1];

		// Update value to 1
		Adj[x][y] = 1;
		Adj[y][x] = 1;
	}
}

// Function to print the created
// Adjacency Matrix
```

```
void printAdjMatrix(int Adj[][N + 1])
{

	// Traverse the Adj[][]
	for (int i = 1; i < N + 1; i++) {
		for (int j = 1; j < N + 1; j++) {

			// Print the value at Adj[i][j]
			printf("%d ", Adj[i][j]);
		}
		printf("\n");
	}
}

// Driver Code
int main()
{

	// Number of vertices
	N = 5;

	// Given Edges
	int arr[][2]= { { 1, 2 }, { 2, 3 },{ 4, 5 }, { 1, 5 } };

	// Number of Edges
	M = sizeof(arr) / sizeof(arr[0]);

	// For Adjacency Matrix
	int Adj[N + 1][N + 1];

	// Function call to create
	// Adjacency Matrix
	createAdjMatrix(Adj, arr);

	// Print Adjacency Matrix
	printAdjMatrix(Adj);

	return 0;
}
```

Q 2. Write a C program for the Implementation of Prim's Minimum spanning tree algorithm.

```
#include <stdio.h>
#include <limits.h>

#define V 5
```

```
nt minKey(int key[], int mstSet[]) {
int min = INT_MAX, min_index;
int v;
 for (v = 0; v < V; v++)
if (mstSet[v] == 0 && key[v] < min)
 min = key[v], min_index = v;
  return min_index;
}

int printMST(int parent[], int n, int graph[V][V]) {
  int i;
  printf("Edge   Weight\n");
  for (i = 1; i < V; i++)
        printf("%d - %d    %d \n", parent[i], i, graph[i][parent[i]]);
}

void primMST(int graph[V][V]) {
  int parent[V]; // Array to store constructed MST
  int key[V], i, v, count; // Key values used to pick minimum weight edge in cut
  int mstSet[V]; // To represent set of vertices not yet included in MST

  // Initialize all keys as INFINITE
  for (i = 0; i < V; i++)
     key[i] = INT_MAX, mstSet[i] = 0;

  // Always include first 1st vertex in MST.
  key[0] = 0; // Make key 0 so that this vertex is picked as first vertex
  parent[0] = -1; // First node is always root of MST

  // The MST will have V vertices
  for (count = 0; count < V - 1; count++) {
     int u = minKey(key, mstSet);
     mstSet[u] = 1;

     for (v = 0; v < V; v++)

        if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v])
           parent[v] = u, key[v] = graph[u][v];
  }

  // print the constructed MST
  printMST(parent, V, graph);
}

int main() {
  /* Let us create the following graph
    2    3
```

```
 (0)--(1)--(2)
  |  / \  |
 6| 8/  \5 |7
  | /    \ |
 (3)-------(4)
  9        */
 int graph[V][V] = { { 0, 2, 0, 6, 0 }, { 2, 0, 3, 8, 5 },
        { 0, 3, 0, 0, 7 }, { 6, 8, 0, 0, 9 }, { 0, 5, 7, 9, 0 }, };

 primMST(graph);

 return 0;
}
```

Q1. Write a C program for the implementation of Topological sorting.
[15 Marks]

```
#include<stdio.h>
#define MAX 200
int n,adj[MAX][MAX];
int front = -1,rear = -1,queue[MAX];
void main() {
        int i,j = 0,k;
        int topsort[MAX],indeg[MAX];
        create_graph();
        printf("The adjacency matrix is:\n");
        display();
        for (i=1;i<+n;i++) {
                indeg[i]=indegree(i);
                if(indeg[i]==0)
                  insert_queue(i);
        }
        while(front<=rear) {
                k=delete_queue();
                topsort[j++]=k;
                for (i=1;i<=n;i++) {
                        if(adj[k][i]==1) {
                                adj[k][i]=0;
                                indeg[i]=indeg[i]-1;
                                if(indeg[i]==0)
                                   insert_queue(i);
                        }
                }
```

```
        }
        printf("Nodes after topological sorting are:\n");
        for (i=0;i<=n;i++)
          printf("%d",topsort[i]);
        printf("\n");
}
create_graph() {
        int i,max_edges,origin,destin;
        printf("\n Enter number of vertices:");
        scamf("%d",&n);
        max_edges = n * (n - 1);
        for (i = 1;i <= max_edges;i++) {
                printf("\n Enter edge %d (00 to quit):",i);
                scanf("%d%d",&origin,&destin);
                if((origin == 0) && (destin == 0)) {
                        printf("Invalid edge!!\n");
                        i–;
                } else
                  adj[origin][destin] = 1;
        }
        return;
}
display() {
        int i,j;
        for (i = 0;i <= n;i++) {
                for (j = 1;jrear) {
                        printf("Queue Underflow");
                        return;
                } else {
                        del_item = queue[front];
                        front = front + 1;
                        return del_item;
                }
        }
        int indegree(int node) {
                int i,in_deg = 0;
                for (i = 1;i <= n;i++)
                  if(adj[i][node] == 1)
                   in_deg++;
                returnin_deg;
        }
```

Write a C program for the implementation of Floyd Warshall's algorithm for finding all pairs shortest path using adjacency cost matrix.

```
/*
 * C Program to find the shortest path between two vertices in a graph
 * using the Floyd-Warshall algorithm
 */

#include <stdio.h>
#include <stdlib.h>

void floydWarshall(int **graph, int n)
{
   int i, j, k;
   for (k = 0; k < n; k++)
   {
      for (i = 0; i < n; i++)
      {
         for (j = 0; j < n; j++)
         {
            if (graph[i][j] > graph[i][k] + graph[k][j])
               graph[i][j] = graph[i][k] + graph[k][j];
         }
      }
   }
}

int main(void)
{
   int n, i, j;
   printf("Enter the number of vertices: ");
   scanf("%d", &n);
   int **graph = (int **)malloc((long unsigned) n * sizeof(int *));
   for (i = 0; i < n; i++)
   {
      graph[i] = (int *)malloc((long unsigned) n * sizeof(int));
   }
   for (i = 0; i < n; i++)
   {
      for (j = 0; j < n; j++)
      {
         if (i == j)
            graph[i][j] = 0;
         else
            graph[i][j] = 100;
      }
   }
   printf("Enter the edges: \n");
```

```
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            printf("[%d][%d]: ", i, j);
            scanf("%d", &graph[i][j]);
        }
    }
    printf("The original graph is:\n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            printf("%d ", graph[i][j]);
        }
        printf("\n");
    }
    floydWarshall(graph, n);
    printf("The shortest path matrix is:\n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            printf("%d ", graph[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Write a C program that accepts the vertices and edges of a graph. Create an adjacency list.

```
// A C Program to demonstrate adjacency list representation of graphs

#include <stdio.h>
#include <stdlib.h>

// A structure to represent an adjacency list node
struct AdjListNode {
    int dest;
    struct AdjListNode* next;
};

// A structure to represent an adjacency liat
```

```
struct AdjList {
   struct AdjListNode *head; // pointer to head node of list
};

// A structure to represent a graph. A graph is an array of adjacency lists.
// Size of array will be V (number of vertices in graph)
struct Graph {
   int V;
   struct AdjList* array;
};

// A utility function to create a new adjacency list node
struct AdjListNode* newAdjListNode(int dest) {
   struct AdjListNode* newNode = (struct AdjListNode*) malloc(
           sizeof(struct AdjListNode));
   newNode->dest = dest;
   newNode->next = NULL;
   return newNode;
}

// A utility function that creates a graph of V vertices
struct Graph* createGraph(int V) {
   struct Graph* graph = (struct Graph*) malloc(sizeof(struct Graph));
   graph->V = V;

   // Create an array of adjacency lists.  Size of array will be V
   graph->array = (struct AdjList*) malloc(V * sizeof(struct AdjList));

   // Initialize each adjacency list as empty by making head as NULL
   int i;
   for (i = 0; i < V; ++i)
       graph->array[i].head = NULL;

   return graph;
}

// Adds an edge to an undirected graph
void addEdge(struct Graph* graph, int src, int dest) {
   // Add an edge from src to dest.  A new node is added to the adjacency
   // list of src.  The node is added at the begining
   struct AdjListNode* newNode = newAdjListNode(dest);
   newNode->next = graph->array[src].head;
   graph->array[src].head = newNode;

   // Since graph is undirected, add an edge from dest to src also
   newNode = newAdjListNode(src);
   newNode->next = graph->array[dest].head;
   graph->array[dest].head = newNode;
```

```
}

// A utility function to print the adjacennncy list representation of graph
void printGraph(struct Graph* graph) {
  int v;
  for (v = 0; v < graph->V; ++v) {
     struct AdjListNode* pCrawl = graph->array[v].head;
     printf("\n Adjacency list of vertex %d\n head ", v);
     while (pCrawl) {
        printf("-> %d", pCrawl->dest);
        pCrawl = pCrawl->next;
     }
     printf("\n");
  }
}

// Driver program to test above functions
int main() {
  // create the graph given in above fugure
  int V = 5;
  struct Graph* graph = createGraph(V);
  addEdge(graph, 0, 1);
  addEdge(graph, 0, 4);
  addEdge(graph, 1, 2);
  addEdge(graph, 1, 3);
  addEdge(graph, 1, 4);
  addEdge(graph, 2, 3);
  addEdge(graph, 3, 4);

  // print the adjacency list representation of the above graph
  printGraph(graph);

  return 0;
}
```

Write a program to sort n randomly generated elements using Heapsort method.

```
#include <stdio.h>
/* function to heapify a subtree. Here 'i' is the
index of root node in array a[], and 'n' is the size of heap. */
void heapify(int a[], int n, int i)
{
    int largest = i; // Initialize largest as root
    int left = 2 * i + 1; // left child
    int right = 2 * i + 2; // right child
    // If left child is larger than root
    if (left < n && a[left] > a[largest])
        largest = left;
    // If right child is larger than root
    if (right < n && a[right] > a[largest])
        largest = right;
    // If root is not largest
    if (largest != i) {
        // swap a[i] with a[largest]
        int temp = a[i];
        a[i] = a[largest];
        a[largest] = temp;

        heapify(a, n, largest);
    }
}
/*Function to implement the heap sort*/
void heapSort(int a[], int n)
{
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(a, n, i);
```

```
    // One by one extract an element from heap
    for (int i = n - 1; i >= 0; i--) {
        /* Move current root element to end*/
        // swap a[0] with a[i]
        int temp = a[0];
        a[0] = a[i];
        a[i] = temp;
         heapify(a, i, 0);
    }
}
/* function to print the array elements */
void printArr(int arr[], int n)
{
    for (int i = 0; i < n; ++i)
    {
        printf("%d", arr[i]);
        printf(" ");
    }
  }
int main()
{
    int a[] = {48, 10, 23, 43, 28, 26, 1};
    int n = sizeof(a) / sizeof(a[0]);
    printf("Before sorting array elements are - \n");
    printArr(a, n);
    heapSort(a, n);
    printf("\nAfter sorting array elements are - \n");
    printArr(a, n);
    return 0;
}
```

Write a C program for the Implementation of Kruskal’s Minimum spanning tree algorithm.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
void main()
{
	clrscr();
	printf("\n\t Implementation of Kruskal's algorithm\n");
	printf("\nEnter the no. of vertices:");
	scanf("%d",&n);
	printf("\nEnter the cost adjacency matrix:\n");
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n;j++)
		{
			scanf("%d",&cost[i][j]);
			if(cost[i][j]==0)
				cost[i][j]=999;
		}
	}
	printf("The edges of Minimum Cost Spanning Tree are\n");
	while(ne < n)
	{
		for(i=1,min=999;i<=n;i++)
		{
			for(j=1;j <= n;j++)
			{
				if(cost[i][j] < min)
```

```
                    {
                        min=cost[i][j];
                        a=u=i;
                        b=v=j;
                    }
                }
            }
            u=find(u);
            v=find(v);
            if(uni(u,v))
            {
                printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
                mincost +=min;
            }
            cost[a][b]=cost[b][a]=999;
        }
        printf("\n\tMinimum cost = %d\n",mincost);
        getch();
}
int find(int i)
{
        while(parent[i])
        i=parent[i];
        return i;
}
int uni(int i,int j)
{
        if(i!=j)
        {
            parent[j]=i;
            return 1;
        }
        return 0;
}
```